10/612831
OCT 28 2003 EXPRESS MAIL NO. EV336610792US

Eur päisches Patentamt
European Patent Office
Office eur péen des brevets

Bescheinigung Certificate Attestation

Die angehefteten Unterlagen stimmen mit der ursprünglich eingereichten Fassung der auf dem nächsten Blatt bezeichneten europäischen Patentanmeldung überein.

The attached documents are exact copies of the European patent application described on the following page, as originally filed.

Les documents fixés à cette attestation sont conformes à la version initialement déposée de la demande de brevet européen spécifiée à la page suivante.

Patentanmeldung Nr. Patent application No. Demande de brevet n°

02425436.9

Der Präsident des Europäischen Patentamts;
Im Auftrag

For the President of the European Patent Office

Le Président de l'Office européen des brevets
p.o.

R C van Dijk

21 x 2

Eur päisches Patentamt
European Patent Office
Office européen des brevets

Blatt 2 der Besch inigung
Sheet 2 of the certificate
Page 2 de l'attestation

Anmeldung Nr.:
Application no.: 02425436.9
Demande n°:

Anmeldetag:
Date of filing: 02/07/02
Date de dépôt:

Anmelder:
Applicant(s):
Demandeur(s):
STMicroelectronics S.r.l.
20041 Agrate Brianza (Milano)
ITALY

Bezeichnung der Erfindung:
Title of the invention:
Titre de l'invention:
A method for executing programs on multiple processors and corresponding processor system

In Anspruch genommene Priorität(en) / Priority(ies) claimed / Priorité(s) revendiquée(s)

Staat: State: Pays:	Tag: Date: Date:	Aktenzeichen: File no. Numéro de dépôt:

Internationale Patentklassifikation:
International Patent classification:
Classification internationale des brevets:
G06F9/38

Am Anmeldetag benannte Vertragstaaten:
Contracting states designated at date of filing: AT/BG/BE/CH/CY/CZ/DE/DK/EE/ES/FI/FR/GB/GR/IE/IT/LI/LU/MC/NL/
Etats contractants désignés lors du depôt:

Bemerkungen:
Remarks:
Remarques:

For the original title, see page 1 of the description

"Procedimento per eseguire programmi su processori e relativo sistema processore"

***

Campo dell'invenzione

La presente invenzione si riferisce ai processori ed ai procedimenti per l'elaborazione dei segnali attuabili con gli stessi.

Descrizione della tecnica nota

Nei sistemi per telefonia cellulare di seconda generazione (ad esempio GSM) o di tipo più evoluto (GPRS, EDGE, UMTS), l'architettura più frequentemente utilizzata consiste di un sistema costituito da due processori. Il primo processore, specializzato nel gestire la parte con più rilevante onere computazionale, è tipicamente costituito da un Digital Signal Processor o DSP. L'altro processore, con compiti di controllo, sincronizzazione ed esecuzione di applicazioni ad alto livello, si configura tipicamente come una CPU.

Un esempio di architettura di questo genere è rappresentata nella figura 1, dove i suddetti processori, indicati rispettivamente come DSP e CPU 1, sono rappresentati insieme alle memorie cache ad essi associate, rispettivamente in funzione di cache istruzioni I$ e di cache dati D$.

Con CMC sono indicati i moduli di interfaccia, denominati Core Memory Controller, che permettono ai due sotto - sistemi facenti capo ai due processori DSP e CPU 1 di interfacciarsi tramite un bus principale B con la memoria principale di sistema MEM e con le varie

unità periferiche P1, P2, P3, P4, ... associate al sistema.

La specifica applicazione nel settore della telefonia cui si è fatto qui riferimento ha peraltro carattere puramente esemplificativo e non implica quindi - neanche in modo indiretto - una limitazione del carattere affatto generale dell'invenzione che sarà descritta nel seguito. Tale invenzione è infatti suscettibile di essere applicata in tutti i campi in cui può essere utile impiegare un microprocessore.

Tornando allo schema della figura 1, la CPU 1 è tipicamente un microprocessore scalare pipelined a 32 bit. Scalare pipelined significa che la sua architettura interna è costituita da diversi stadi logici, ognuno dei quali contiene un'istruzione in un ben specifico stato. Tale stato può essere di:

- caricamento dell'istruzione stessa dalla memoria,
- decodifica dell'istruzione,
- indirizzamento di un file di registri,
- esecuzione,
- scrittura/lettura dati dalla memoria.

Il numero di bit su cui opera la CPU 1 si riferisce all'ampiezza dei dati sul quale la macchina opera. Le istruzioni sono generate ed eseguite una alla volta, in uno specifico ordine definito tramite compilazione.

L'altro processore, indicato con DSP, è tipicamente un microprocessore superscalare o VLIW (acronimo per Very Long Instruction Word) pipelined a 128 bit.

Superscalare pipelined significa che la sua architettura interna è costituita da diversi stadi

logici alcuni dei quali possono eseguire istruzioni in parallelo, ad esempio nella fase di esecuzione. Tipicamente, il parallelismo è di 4 istruzioni ciascuna (pari a 128 bit) mentre i dati sono espressi in 32 bit.

Il processore si dice superscalare se le istruzioni sono riordinate dinamicamente in fase di esecuzione in modo da alimentare gli stadi di esecuzione che potenzialmente possono lavorare in parallelo, anche alterando l'ordine generato staticamente dalla compilazione del codice sorgente, se le istruzioni non presentano dipendenze mutue. Lo svantaggio principale di questo approccio sta nella complessità della macchina risultante, in cui la logica di schedulazione delle istruzioni può risultare essere una delle parti più rilevanti in termini di numero di porte.

Si parla di processore VLIW se le istruzioni sono riordinate staticamente in fase di compilazione ed eseguite in quell'ordine fisso, non modificabile in fase di esecuzione. Il vantaggio di tale approccio è che si elimina tutta la logica di gestione della schedulazione poiché si esegue questo task in sede di compilazione.

Lo svantaggio principale sta nel fatto che il codice compilato è strettamente dipendente dall'implementazione della macchina su cui viene eseguito. Ad esempio, pur a parità di architettura di insieme di istruzioni (Istruction Set Architecture o ISA), una macchina con N unità di esecuzione non può eseguire un codice compilato per una macchina con K unità di esecuzione se K non è uguale a N. Da ciò consegue che non c'è "compatibilità binaria" tra diverse generazioni di processori con lo stesso ISA.

Si ricorda che per compatibilità binaria si intende la proprietà esistente tra un gruppo di processori ognuno dei quali è in grado di eseguire uno stesso dato codice macchina binario.

Non si possono altresì creare sistemi multiprocessori (ciascuno con diverso numero di unità di esecuzione) suscettibili di scambiarsi processi in corso di esecuzione.

Nello schema della figura 1, ogni processore possiede la propria cache dati D$ e la propria cache istruzioni I$, in modo da poter caricare dalla memoria principale MEM sia i dati sui quali operare, sia le istruzioni da eseguire in parallelo. Poiché i due processori CPU1 e DSP sono collegati alla memoria principale MEM attraverso il bus di sistema B i due processori si trovano tipicamente a competere per l'accesso a tale memoria quando un'istruzione e/o i dati su cui devono operare devono essere localizzati nella memoria principale non essendo disponibili nelle proprie cache.

Un sistema ispirato all'architettura rappresentata nella figura 1 ha una partizione di lavoro e di processi rigida e non modificabile, tale da rendere asimmetrico il carico di lavoro ed i programmi software da eseguire.

A titolo di riferimento, un processore quale la CPU 1 possiede di solito 16 Kbyte di cache e 16 Kbyte di cache istruzioni. Questo mentre il processore DSP possiede di solito 32 Kbyte di cache dati e 32 Kbyte di cache istruzioni.

Il diagramma di flusso della figura 2 illustra lo schema logico della CPU descritto dall'alto verso il basso. Il primo stadio, indicato con 10, genera

l'indirizzo di memoria al quale è associato l'istruzione da eseguire, tale indirizzo è detto Program Counter. Lo stadio 10 si configura quindi tipicamente come uno stadio di fetch, mentre l'istruzione così caricata viene decodificata nello stadio 12 separando il campo di bit che ne definisce la funzione (ad esempio di somma di 2 valori contenuti in due registri localizzati nel registro file) rispetto ai campi di bit che indirizzano gli operandi. Tali indirizzi sono inviati ad un file di registri dal quale (in uno stadio indicato con 14) vengono letti gli operandi dell'istruzione. Gli operandi ed i bit che definiscono la funzione da eseguire sono inviati all'unità di esecuzione che, in uno stadio 16, realizza l'operazione desiderata, ad esempio l'operazione di somma a cui si è fatto riferimento in precedenza. Il risultato può quindi essere ri-memorizzato nel file di registri in uno stadio 18 correntemente denominato stadio di Write Back.

Il processo schematicamente rappresentato dalla figura 2 opera in unione ad un unità di load/store che consente di leggere/scrivere eventuali dati in memoria con l'ausilio di specifiche istruzioni dedicate allo scopo.

E' immediato riconoscere il fatto che il set di istruzioni è in corrispondenza biunivoca con una data architettura di CPU di microelaborazione.

Il diagramma di flusso della figura 3 mostra invece lo schema logico del processore DSP. Anche in questo caso è previsto uno stadio iniziale di fetch 20 cui è associato logicamente in cascata uno stadio 20a di emissione delle istruzioni. Il riferimento numerico 22 indica invece uno stadio di decodifica mentre il

riferimento 24 indica un file registri (si faccia riferimento agli stadi 14 e 16 della figura 2). Il riferimento 28 indica uno stadio di ri-memorizzazione nel file registri complessivamente affine allo stadio 18 della figura 1. Nello schema della figura 3 il riferimento 26 indica collettivamente una pluralità di stadi di esecuzione suscettibili di essere eseguiti in parallelo.

Tanto nella figura 1 quanto nella figura 3 il riferimento CW indica le linee di diramazione delle parole di controllo.

Si apprezzerà che la principale differenza fra lo schema della figura 2 e lo schema della figura 3 è data dal fatto che nello schema della figura 3 è prevista la possibilità di lavorare in parallelo su diversi insiemi di istruzioni. Un'altra differenza sta nel fatto che lo schema della figura 3 prevede l'impiego di un maggior numero di unità esecutive disponibili suscettibili di operare in parallelo in un processore superscalare e VLIW. In entrambi i casi il set di istruzioni sta in corrispondenza biunivoca con una data architettura di microelaborazione.

Supponendo che i due insiemi di istruzioni destinati ad essere eseguite dai processori CPU 1 e DSP siano diversi fra loro (così come comunemente si riscontra nell'architettura di processori wireless) si comprende che istruzioni (e quindi task da eseguire) suscettibili di essere eseguiti dal processore CPU 1 non possono essere eseguiti dal processore DSP e viceversa.

Affinché ciò sia possibile è necessario compilare ogni processo per ogni processore, aumentando così la memoria del programma. Ogni volta che un processo deve

essere eseguito da uno specifico processore, bisogna poi caricare ed eseguire il codice di quel task che è stato compilato per quel processore. Si riscontra inoltre il problema legato al fatto di dovere correlare i diversi punti di esecuzione parziale dei programmi quando li si voglia spostare da un processore all'altro (ovvero rimappare correttamente i Program Counter), di dovere convertire tutti i dati di elaborazione dal sistema di rappresentazione di un processore all'altro (ad esempio il contenuto dei registri di stato e general purpose).

Questi problemi sono di difficile risoluzione, per cui in genere un processo viene compilato ed eseguito su un solo processore.

Con riferimento alle figure 4 e 5 è possibile considerare una sequenza di gruppi di istruzioni di detti processi.

In generale si distinguono due tipi di processo, ossia:

- quelli relativi al sistema operativo e ad applicazioni che usano chiamate a funzioni del sistema operativo, e

- quelli relativi all'elaborazione di contenuti multimediali (audio/video/grafica).

In modo specifico, nello schema della figura 4 i riferimenti OsTask 1.1, 1.2, etc. illustrano processi suscettibili di essere eseguiti dal processore CPU1. I processi indicati con MmTask2.1, MmTask2.2, MmTask2.3, ... identificano invece processi compilati in modo da essere eseguiti dal processore DSP.

A partire dallo schema della figura 4, che illustra una possibile assegnazione dei task ai due processori, si può immediatamente risalire allo schema

della figura 5 che illustra il corrispondente flusso di istruzioni.

Posto pari a cento il tempo totale di esecuzione dei processi, si osserva che i primi processi tipicamente durano il 10% del tempo, mentre i secondi occupano una parte ben più rilevante, pari al 90%.

Ancora, i primi processi contengono istruzioni generate dal compilatore del processore CPU 1 e quindi possono essere eseguiti dallo stesso, ma non dal processore DSP. Per i secondi processi vale il discorso esattamente complementare, nel senso che contengono istruzioni generate dal compilatore del processore DSP e possono essere quindi eseguiti da tale processore, ma non dall'altro processore CPU 1.

Si osserva ancora che il processore CPU 1 è caratterizzato da un proprio flusso di compilazione, indipendente e distinto da quello del processore DSP.

Dato il modesto carico di lavoro, si evince che il processore CPU 1 potrebbe essere anche spento quando non in uso, consentendo un sensibile risparmio energetico.

Questa ipotetica soluzione (spegnimento del processore CPU 1 quando non utilizzato) si scontra con il fatto che le relative procedure di spegnimento o power-down introducono latenze aggiuntive di elaborazione e si sommano al valore di 10% indicate in precedenza. Le suddette procedure prevedono infatti:

- lo spegnimento del processore CPU1, ad eccezione del rispettivo file registri tramite gating del segnale di clock che alimenta tutti i registri interni,

- lo spegnimento completo del processore CPU, salvo il fatto che viene mantenuta l'alimentazione energetica delle memorie cache e

- lo spegnimento del CPU nel suo complesso, comprese le cache dati ed istruzioni.

Tuttavia, dato che lo stato del singolo processore deve essere ripristinato durante la riaccensione a seguito di una delle operazioni citate in precedenza, le latenze introdotte variano da decine di microsecondi a decine/centinaia di millisecondi. Queste latenze risultano particolarmente dispendiose, sia dal punto di vista energetico, sia dal punto di vista computazionale.

Infine, il processore DSP è costretto a lavorare circa al 90% della sua capacità computazionale. Ciò implica un evidente asimmetria nel carico di lavoro dal processore CPU rispetto al processore TSP, asimmetria che si rivela anche negli algoritmi di gestione dell'alimentazione (power-management), che sono distinti per i due processori.

Scopi e sintesi dell'invenzione

La presente invenzione si prefigge lo scopo di fornire una soluzione in grado di superare gli inconvenienti delineati in precedenza.

Secondo la presente invenzione, tale scopo viene raggiunto grazie ad un procedimento avente le caratteristiche richiamate in modo specifico nelle rivendicazioni che seguono. L'invenzione riguarda anche il corrispondente sistema processore, in particolare multiprocessore.

La presente invenzione offre quindi una soluzione che consente:

- di eseguire programmi in modo indistinto su due o più processori, ad esempio del tipo VLIW, costituenti un sistema in condizioni di lavoro dinamicamente variabili,

- assicurare la compatibilità binaria fra due o più processori, in particolare processori VLIW aventi lunghezza massima diversa di Long Instruction Word.

Tutto questo facendo si che per supportare l'esecuzione dei processi non sia necessario l'uso di un'architettura multi-processing asimmetrica.

In sostanza, la soluzione secondo l'invenzione consente di realizzare un'architettura di multielaborazione o multi-processing comprendente una pluralità di processori.

Nell'esempio di applicazione descritto nel dettaglio nel seguito (esempio che si rammenta essere tale) i processori considerati sono tutti del tipo VLIW. La soluzione secondo l'invenzione è però applicabile, ad esempio, ad architetture comprendenti uno o più processori VLIW in combinazione con almeno un processore superscalare, l'unico aspetto importante essendo che i processori in questione abbiano la stessa architettura di insieme di istruzioni (ISA).

La soluzione secondo l'invenzione è infatti basata sul riconoscimento di alcuni fatti essenziali.

Si considerino in primo luogo due o più processori (per semplicità si farà nel seguito riferimento solo a processori VLIW) aventi la stessa architettura di insieme di istruzioni (ISA), il che implica che su ciascuno di essi possa essere eseguita una sequenza lineare di istruzioni di base. Ciascun processo/task è stato compilato per un certo processore VLIW come flusso di istruzioni lunghe in cui la più lunga ha una dimensione corrispondente al parallelismo a livello di istruzioni per cui il processore VLIW è stato progettato.

Ciascun processore VLIW può eseguire istruzioni con una lunghezza massima diversa: ad esempio, il primo può eseguire al massimo 4 istruzioni base in parallelo

per ciascun ciclo di clock, mentre il secondo può utilizzare al massimo 8 istruzioni in parallelo per ciascun ciclo di clock.

Ciascun processore ha un'unità di emissione delle istruzioni che tipicamente legge dalle cache di istruzioni un numero di istruzioni pari a quello che è in grado di trattare in parallelo. Ad esempio, se il processore è in grado di trattare in parallelo fino a 4 istruzioni, la relativa unità di emissione delle istruzioni può leggere simultaneamente fino a 4 istruzioni ma non un numero maggiore.

E' possibile prevedere per un secondo processore VLIW un'unità di istruzioni modificate che è in grado di leggere all'ingresso parole corrispondenti ad istruzioni molto lunghe originariamente compilate in vista dell'esecuzione su un primo processore VLIW e di emettere in uscita parole corrispondenti ad istruzioni molto lunghe pronte per essere eseguite sul secondo processore VLIW.

Questo cambiamento della lunghezza delle istruzioni può essere effettuato in condizioni di run time senza alcuna esigenza di ricompilazione e di duplicazione del codice: ciascuna istruzione nella parola di istruzione molto lunga è stata infatti schedulata dal compilatore in modo da essere indipendente rispetto alle altre. Questo risultato è ottenuto sotto forma di una cascata di semplici operazioni di suddivisione (split) e di inserimento di istruzioni fittizie (no-operation) in funzione della lunghezza delle istruzioni di un qualunque processore VLIW.

Per consentire la ri-allocazione dinamica di un processo compilato ed eseguito su un primo processore VLIW in modo da eseguirlo su un secondo processore VLIW, nella memoria del sistema si mantiene una tabella

in cui è immagazzinata l'informazione relativa a ciascun processo in termini identificativi (si può trattare di un semplice numero d'ordine) del processore VLIW per cui è stato compilato nonché un identificativo (anche in questo caso si può trattare semplicemente di un numero) dell'ultimo processore VLIW sul quale, nell'ambito della pluralità di processori comprendente l'architettura multi-processore, il processo è stato eseguito, nonché - naturalmente - le coordinate per rintracciare il suo contesto nella memoria principale.

Per contesto di un processore si intende qui indicare essenzialmente:

- il valore del Program Counter,
- il contenuto del file registri (register file),
- la memoria stack dei dati.

Utilizzando la soluzione secondo l'invenzione è possibile conseguire una notevole semplificazione in termini di ambiente di programmazione e di esigenze di memoria e di assorbimento di potenza. Inoltre, la soluzione secondo l'invenzione risolve un problema rilevante nell'ambito dei processori VLIW, vale a dire quello della compatibilità binaria fra architetture e parallelelismi diversi, consentendo ad esempio il trasferimento dinamico del carico di lavoro di processo fra processori VLIW simmetrici dal punto di vista dell'insieme delle istruzioni ed asimmetrici in termini di parallelismo di istruzione.

Breve descrizione dei disegni annessi

L'invenzione sarà ora descritta, a puro titolo di esempio non limitativo, con riferimento ai disegni annessi, nei quali:

- le figure 1 a 5, relative alla tecnica nota, sono già stati descritte in precedenza,

- la figura 6 illustra, sotto forma di uno schema a blocchi, l'architettura di un sistema multiprocessore suscettibile di operare secondo l'invenzione,

- le figure 7 e 8 illustrano in maggior dettaglio i criteri di trattamento di processi nell'ambito di un'architettura operante secondo l'invenzione,

- la figura 9 illustra i criteri di organizzazione di un'unità per l'emissione di istruzioni operanti secondo l'invenzione,

- le figure 10 a 12 illustrano dettagli di attuazione di vari interventi sul set di istruzioni nell'ambito di un sistema operante secondo l'invenzione, anche in relazione alla posizione in cui si svolge la formattazione delle istruzioni, e

- la figura 13 illustra la struttura di una tabella suscettibile di essere implementata nell'ambito dell'invenzione.

L'osservazione dello schema a blocchi della figura 6 permette di rendersi conto che tale schema riproduce essenzialmente l'impostazione generale dello schema della figura 1, già considerata in precedenza.

A differenza dell'architettura rappresentata nella figura 1 (che è un'architettura intrinsecamente asimmetrica, per la presenza dei due processori CP1 e DSP, con caratteristiche diverse), lo schema della figura 6 è intrinsecamente simmetrico, in quanto prevede la presenza di due (o più) processori di tipo VLIW - qui indicati rispettivamente con VLIW1 e VLIW2 - che, pur essendo istanziati secondo criteri diversi, operano in condizioni di sostanziale simmetria, in quanto essi sono in grado di eseguire gli stessi processi, senza che ciò richieda una ricompilazione ovvero una duplicazione dei codici oggetto per i due processori. Tutto questo peraltro potendosi benissimo

ammettere la presenza di due (o più) processori VLIW aventi un diverso parallelismo hardware disponibile.

In particolare, con riferimento alle figure 7 e 8, si consideri in una prima fase di compilare il codice sorgente di un processo detto Task1 relativo al sistema operativo, da eseguire sul processore LVIW1 (si fa riferimento allo schema della figura 6) e con il relativo compilatore. Si supporrà altresì che al più tale processore possa eseguire in parallelo 4 istruzioni per ogni ciclo di clock.

Inoltre, si consideri nella stessa fase di compilare il codice sorgente di un processo detto Task2, relativo ad esempio all'applicazione audio/video/grafica multimediale, da eseguire (sempre con riferimento allo schema di figura 6) sul processore VLIW2, e con il relativo compilatore Si supporrà altresì che al più il processore VLIW2 possa eseguire 8 istruzioni in parallelo per ogni ciclo di clock.

Naturalmente i suddetti valori (4 e 8, rispettivamente) ed il fatto di riferirsi a - due - processori VLIW rispondono a pure esigenze esemplificative, senza in alcun intento limitativo la portata dell'invenzione.

Si ricorda ancora il fatto che i due processori VLIW hanno la stessa architettura di set di istruzioni (ISA). Ciò significa che sono definiti dallo stesso elenco di istruzioni, sintassi e semantica e lunghezza (ad esempio 32 bit per istruzione base).

In figura 8 si mostra come si voglia associare staticamente e dinamicamente ognuno dei processi in modo indifferente su ognuno dei due processori VLIW1 e VLIW2 in base a decisioni che possono cambiare dinamicamente. Si può trattare, ad esempio, di decisioni basate sul carico dinamico di ognuno dei processori, sulla frequenza operativa dinamica e sul

consumo di energia istantaneo. Questo con lo scopo finale di suddividere equamente il carico di lavoro sui due processori.

Ancora, si rileva che ciascuno dei due (o più) processori considerati è in grado di funzionare con una frequenza di lavoro diversa, per cui ciascun processore può dinamicamente cambiare la frequenza operativa senza compromettere il suo corretto funzionamento.

Ad esempio, il Task1, così come mostrato in figura 7, è compilato per generare un flusso di istruzioni lunghe con massima lunghezza 4, supponendo di eseguirle su processore VLIW1.

Il Task 2 è invece compilato per generare un flusso di istruzioni lunghe con massima lunghezza 8, supponendo di volerla eseguire su processore VLIW2.

Sempre a titolo esemplificativo si supponga ancora di voler invertire l'associazione definita, volendo associare i Task2 al processore VLIW1 ed il Task1 al processore VLIW2.

La figura 9 mostra come il cosiddetto istruction issue unit (ovvero l'unità di emissione delle istruzioni, nel seguito indicata semplicemente come IIU) di un processore come notato in figura 3 deve essere modificata per assolvere il seguente compito al fine di assicurare la compatibilità binaria senza duplicare il codice oggetto.

Si supponga che la IIU proposta dal processore VLIW1 riceva istruzioni lunghe 8×32 bit mentre il processore può eseguirne al più 4×32 bit per ciclo.

L'unità IIU dovrà quindi suddividere (splittare) le istruzioni in ingresso in due istruzioni lunghe 4×bit.

Si supponga ora che l'unità IIU del processore VLIW2 riceva istruzioni lunghe 4×32 bit mentre il processore può eseguirne al più 8×32 bit per ciclo.

L'unita IIU in questione deve quindi allungare l'istruzione di ingresso mediante 4x32 bit istruzioni nulle del tipo no-operation o, in breve, nop.

Un tipico esempio (ben noto agli esperti del settore) di un istruzione nop è l'istruzione di effettuare la somma del numero 0 con se stesso da scrivere nel registro che contiene tale 0, che è a sola lettura. Si tratta quindi di un'istruzione che esprime un non senso voluto. Inoltre le istruzioni nop provocano lo spegnimento di quattro su otto unità funzionali riducendo il consumo di potenza del processore VLIW2.

Gli esempi sopra considerati possono evidentemente essere generalizzati nel modo seguente.

1. Si consideri essere L1 la lunghezza massima dell'istruzione lunga che il processore VLIW1 consente di eseguire: ad esempio VLIW1 può eseguire fino a 13 istruzioni in parallelo, ognuna con lunghezza pari a 32 bit.

2. Si consideri essere L2 la lunghezza massima dell'istruzione lunga che il processore VLIW2 consente di eseguire: ad esempio VLIW2 può eseguire fino a 3 istruzioni in parallelo ciascuna con lunghezza 32 bit.

3. Sia L1 >=L2.

4. Sia A=L1/L2 il risultato intero della divisione fra L1 e L2. Per semplicità, si supponga che tale risultato sia pari a 4.

5. Sia B il resto intero dell'operazione L1/L2, resto che è evidentemente minore di L2. Si può supporre, ad esempio che tale resto sia pari ad 1.

6. Si vuole eseguire un'istruzione di lunghezza L1 sul processore VLIW2.

Di conseguenza:

- se B è uguale a zero, si decompone la lunghezza L1 in A istruzioni lunghe L2,

- se B è diverso da zero, si decompone L1 in A istruzioni lunghe L2, alle quali si aggiunge una ulteriore istruzione lunga L2, composta dalle rimanenti istruzioni di L1 non utilizzate per comporre le precedenti A istruzioni lunghe L2, in numero di B istruzioni del set L1, aggiungendo ancora L2-B istruzioni nop.

7. Si vuole eseguire un'istruzione di lunghezza L2 sul processore VLIW1; all'istruzione L2 si devono quindi aggiungere L1-L2 istruzioni nop.

Si apprezzerà che gli stessi criteri si applicano se L2>L1, essendo sufficiente scambiare l'indice 1 con l'indice 2 senza compromettere la generalità del metodo.

La figura 9 illustra un ulteriore esempio di quanto detto sopra, esempio che si riferisce in modo specifico al caso in cui L1 è pari a 4 e L2 è pari a 8.

Un'altra soluzione, cui fanno specifico riferimento le figure 10 e 11, può consistere nell'esplicitare nelle istruzioni macchina del processore diversi bit cosiddetti di fine gruppo o "stop bundle" relativi a diverse lunghezze di parola, cioè diversi livelli massimi di parallelismo eseguibili dalla macchina.

Nel caso della figura 10, un processore costruito con parallelismo A utilizza come indicatore di fine bundle il bit 31 ignorando il bit 30. Viceversa, un processore costruito con parallelismo B utilizza come indicatore di fine bundle il bit 30, ignorando il bit 31. La verifica della fine di bundle viene fatta normalmente a livello di instruction issue (IIV).

Poichè ogni processore ha una cache istruzioni I$ associata e (potenzialmente) un compressore/decompressore del codice da eseguire, la

soluzione illustrata nella figura 10 può essere generalizzata nel modo seguente.

Sia dato un generico istruction set con istruzioni codificate su N bit (ad esempio, sia N=32). Si supponga altresì che uno di questi bit codifichi l'istruzione di stop bundle per il processore. A queste parole si aggiungono K bit che codificano l'informazione di stop bundle per diverse lunghezze di parola. Ad esempio, sia K=4, per cui i 4 bit in questione codificano l'informazione stop bundle per lunghezze rispettivamente pari a 2, 4 , 6, 8.

In fase di decodifica, ossia in fase di ririempimento (refill) della cache, ovvero in fase di decompressione del codice, può essere effettuata l'operazione di collocamento dell'opportuno stop bundle nel bit di istruzione "effettiva" a seconda del parallelismo della macchina che esegue il codice. Si veda al riguardo la figura 11.

In particolare, in tale figura si vede - nella parte superiore, indicata con A - una situazione in cui il bit 31 non ha previsto l'inserimento di un'informazione di stop bundle. Tutto questo per quanto rigurda il bit 31 della porzione dell'istruzione, indicata con AI, in cui è inserita l'istruzione, vera e propria. Nell'appendice di istruzione, indicata con IA, sono invece inserite le informazioni di stop bundle ed in particolare per ampiezze di uscita rispettivamente pari a 2 (SB2), a 4 (SB4), a 6 (SB6) e ad 8 (SB8), rispettivamente.

La situazione rappresentata nella parte B della figura 11 si riferisce invece alla esecuzione su una macchina con larghezza 4. In tale situazione, lo stop bundle per la larghezza pari a 4 viene collocato nell'istruzione in corrispondenza del bit 31. Tutto questo con la conseguente possibilità di scartare la

parte di appendice IA, così da dare origine ad un formato di istruzione rappresentato nella parte C della stessa figura 11. Il modulo che realizza l'operazione appena descritta viene definito modulo di formattazione delle istruzioni o "instruction formatter.

L'operazione appena descritta può essere effettuata in diverse posizioni nel sistema. Si osserverà ancora che nella figura 10 sono state rappresentate, procedendo dall'alto verso il basso, le diverse possibili combinazioni dei bit in posizione 30 e 31, destinati a fungere rispettivamente come stop-bundle per le larghezze di uscita B (bit 30) ed A (bit 31) i 4 seguenti casi:

- assenza della fine del bundle,
- fine del bundle A,
- fine del bundle B, e
- fine di entrambi i bundle A e B.

Osservando la figura 12, si può vedere che è possibile collocare il suddetto formattatore, indicato con IF tra l'eventuale decompressore di istruzioni CD e - in ogni caso - fra la memoria principale MM e la cache I$. Alternativamente, il formattatore IF può essere posizionato subito prima del processore VLIW e quindi dopo la cache I$ o addirittura nello stadio di decodifica del processore VLIW. Quest'ultimo caso è lo stesso descritto precedentemente con riferimento alla figura 10 per K=2, N=30.

In fase di compilazione, quindi, il compilatore genera tutte le informazioni aggiuntive (stop bundle) per tutte le larghezze di uscita previste nella appendice di istruzione IA. Queste saranno poi utilizzate dal formattatore IF a seconda della configurazione dello stesso.

L'operazione svolta dal formattatore IF consiste solo nell'inserzione di un bit e nell'eliminazione

dell'appendice IA. Si tratta di operazioni suscettibili di essere implementate in modo elementare a livello hardware, così come evidente alle persone esperte del settore.

Le istruzioni che compongono i task della figura 8 sono presenti nella memoria del sistema MEM e sono indirizzate mediante il Program Counter provvisto in ciascun processore VLIW provvisto (vedere la figura 6). Tali istruzioni sono caricate dall'unità fetch di figura 3 ed opportunamente adattate al parallelismo del processore secondo la soluzione implementata dall'unità IIU descritta in precedenza con particolare riferimento alla figura 9.

La soluzione secondo l'invenzione offre il vantaggio rilevante dato dalla completa compatibilità binaria fra i due (o più) processori, intesa come capacità di eseguire indistintamente detti processi sui vari processori VLIW compresi in un sistema multiprocessore utilizzando lo stesso codice compilato senza inutili duplicazioni. Tale capacità consente inoltre di distribuire dinamicamente il carico computazionale sui vari processori in modo da poter equalizzare la frequenza operativa dei processori stessi rispetto al massimo punto. Si consegue così un risparmio di potenza dissipata che, come è noto, dipende linearmente dalla frequenza operativa del processore.

Al fine di meglio chiarire come sia possibile spostare l'esecuzione di un processo da un processore VLIW all'altro si consideri l'esistenza di una tabella memorizzata nella memoria MEM del sistema.

Con riferimento alla figura 13, descrivendola da sinistra verso destra, la tabella mostra:

- una lista di processi (Process) in esecuzione o sospesi su un qualsivoglia processore,

- il numero progressivo (Num) dello stesso in base all'ordine di attivazione,

- la percentuale (% VLIW) di potenza massima del processore che è utilizzata da detto processo,

- il tempo di esecuzione (Exec.time) che, se nullo indica che il processo è temporaneamente sospeso dall'esecuzione,

- la quantità di memoria (Memory) del sistema utilizzato dal processo per poter eseguire la funzione al quale è preposto,

- la lunghezza massima (Compiled For VLIW Of Length) dell'istruzione lunga che il processore VLIW può eseguire e per il quale era stata generata durante la compilazione,

- lunghezza massima (Execution on VLIW of Length) dell'istruzione lunga del processore VLIW sul quale è eseguito, e

- l'indirizzo della porzione di memoria (Memory address) nel quale i dati e le istruzioni sono memorizzate, in altre parole il contesto.

Questa tabella è accessibile da un processo detto di controllo che è eseguito per un tempo prefissato su ognuno dei processori VLIW. Attraverso tale processo, ogni processore ha quindi la possibilità di consultare e aggiornare la tabella al fine di equalizzare il proprio carico di lavoro rispetto al secondo processore. Tale tabella contiene altresì le coordinate affinché un processore possa prendere possesso ed eseguire uno dei processi elencati.

La soluzione descritta è evidentemente estendibile ad un numero qualsiasi di processori VLIW che costituiscono il sistema ed ogni dei quali abbia una qualunque lunghezza massima dell'istruzione lunga da eseguire, ove detta lunghezza vari da un processore all'altro.

Naturalmente, fermo restando il principio dell'invenzione, i particolari di realizzazione e le forme di attuazione potranno essere ampiamente variati rispetto a quanto descritto ed illustrato senza per questo uscire dall'ambito della presente invenzione, così come definito dalle rivendicazioni annesse.

Fig_1

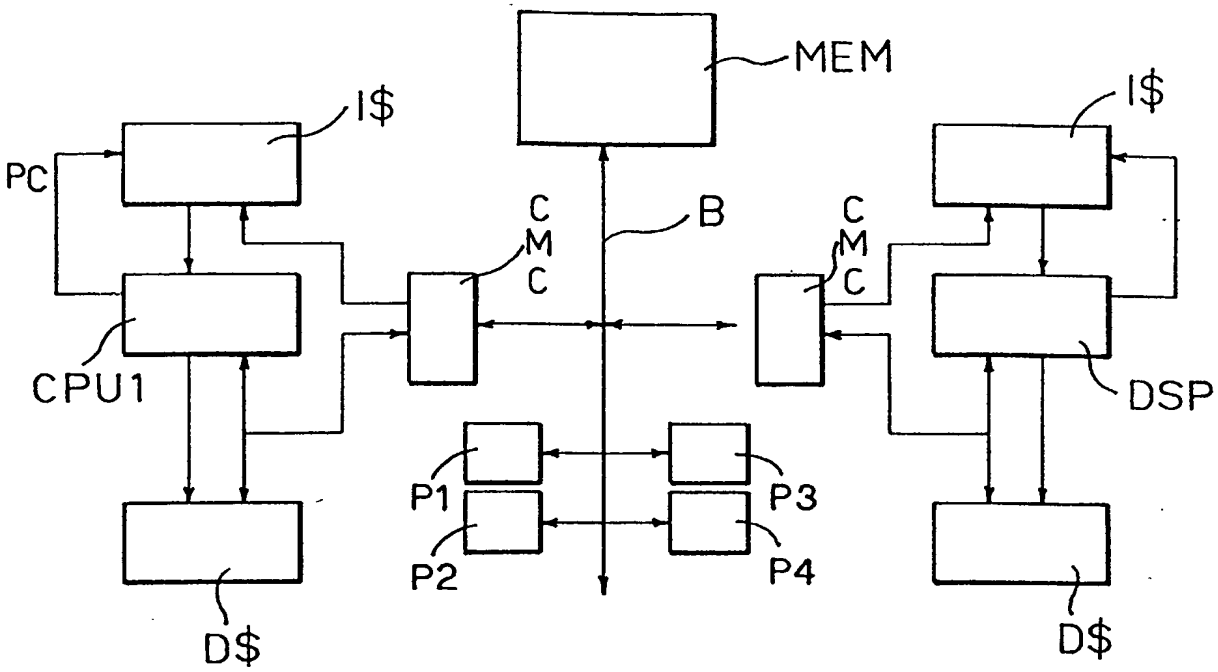
MEM
I$
PC
C
M
C
B
C
M
C
I$
CPU1
DSP
P1
P3
P2
P4
D$
D$

Fig_2

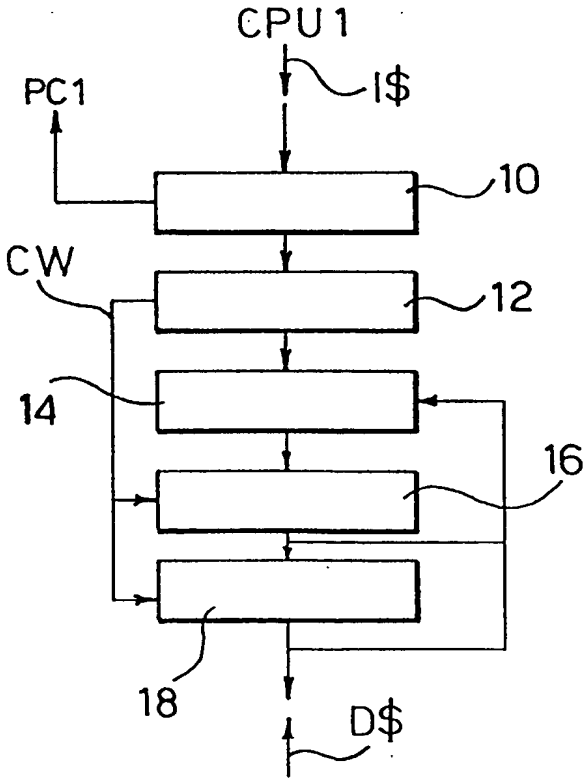
CPU1
I$
PC1
10
CW
12
14
16
18
D$

Fig_3

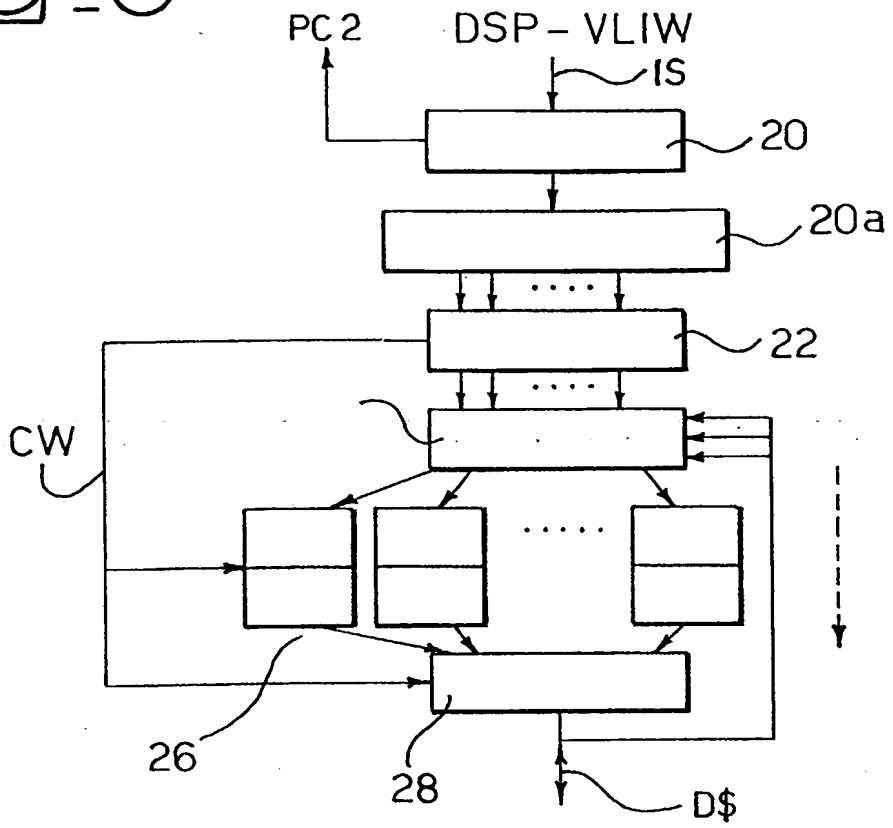
PC2
DSP – VLIW
IS
20
20a
22
CW
26
28
D$

Fig_4

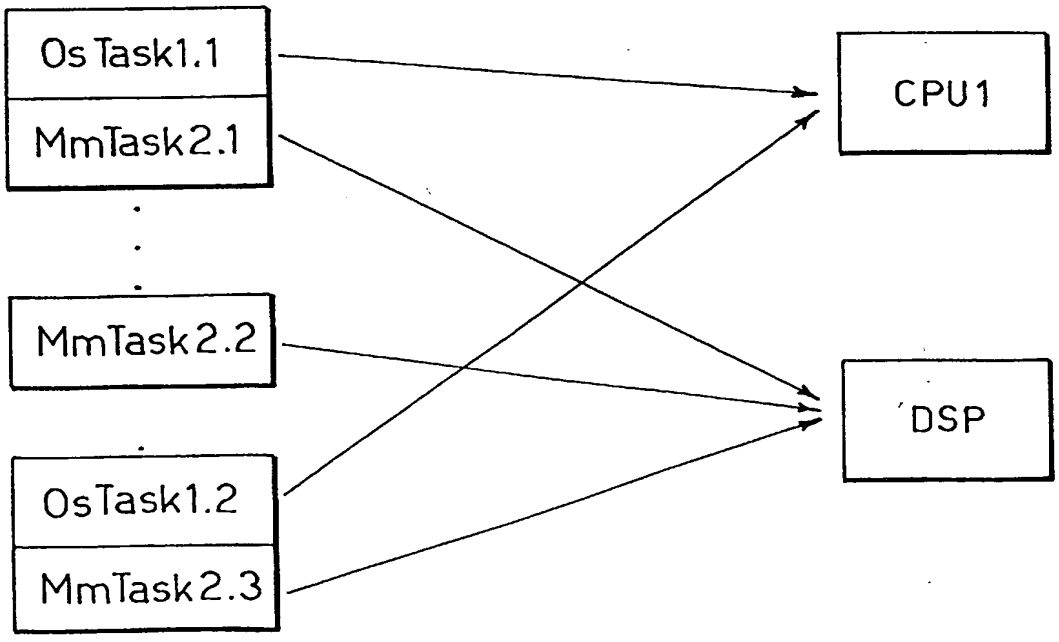
Os Task1.1
MmTask2.1
MmTask2.2
OsTask1.2
MmTask2.3
CPU1
DSP

Fig_5

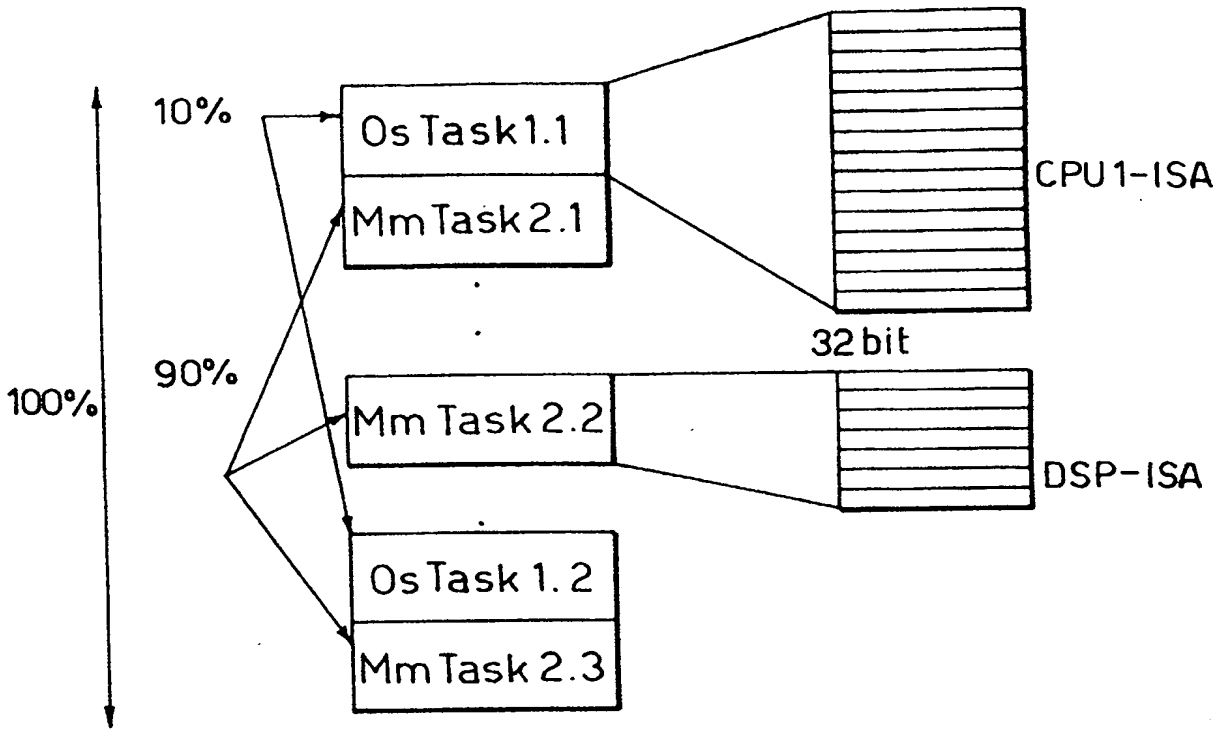
10%
90%
100%
Os Task 1.1
Mm Task 2.1
Mm Task 2.2
Os Task 1.2
Mm Task 2.3
CPU 1-ISA
32 bit
DSP-ISA

Fig_6

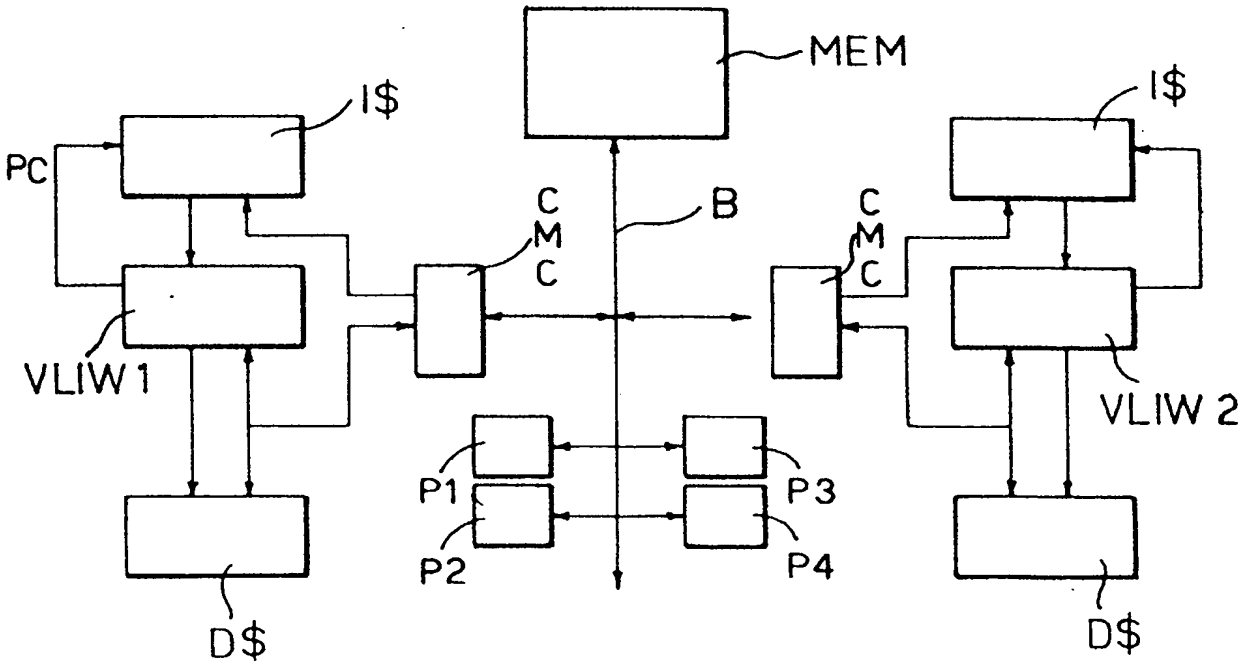
MEM
I$
PC
C
M
C
B
C
M
C
I$
VLIW 1
VLIW 2
P1
P2
P3
P4
D$
D$

Fig_7

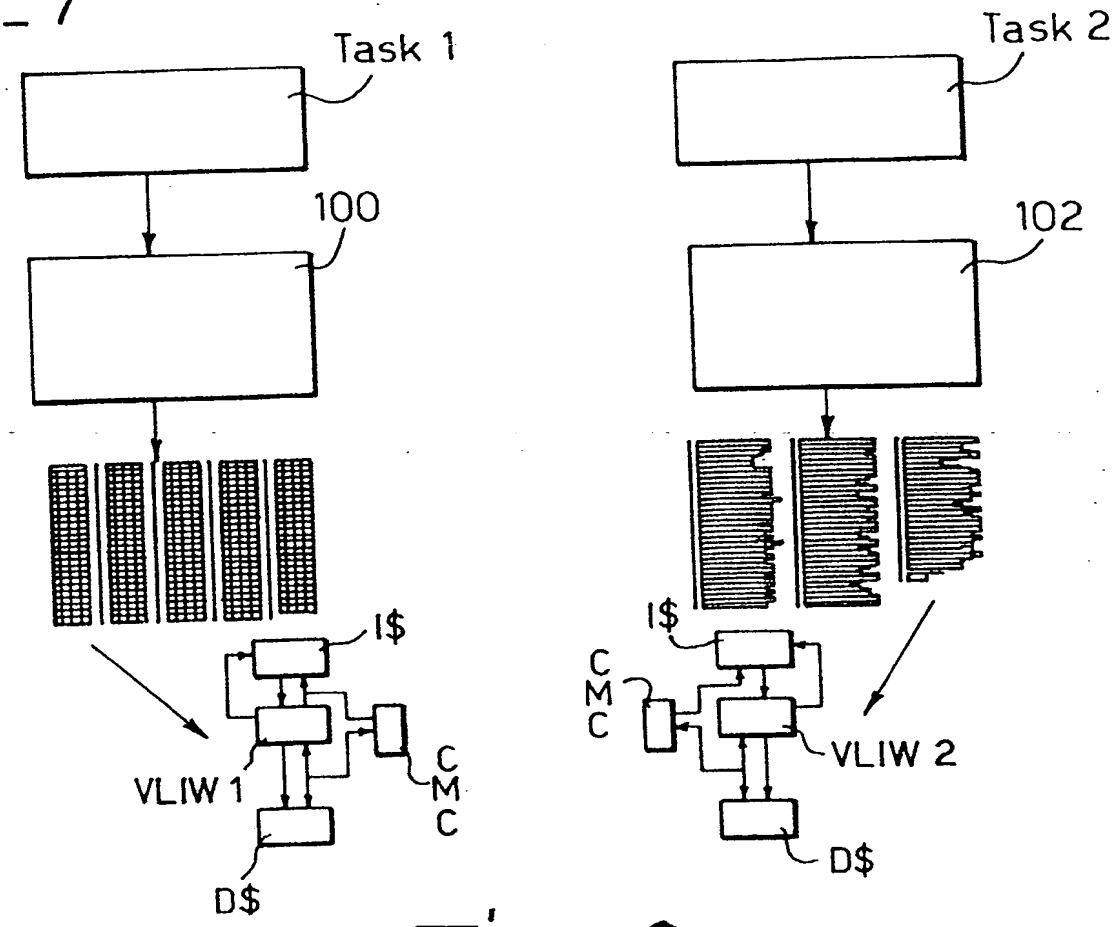
Task 1
100
I$
VLIW 1
C
M
C
D$
Task 2
102
I$
C
M
C
VLIW 2
D$

Fig_8

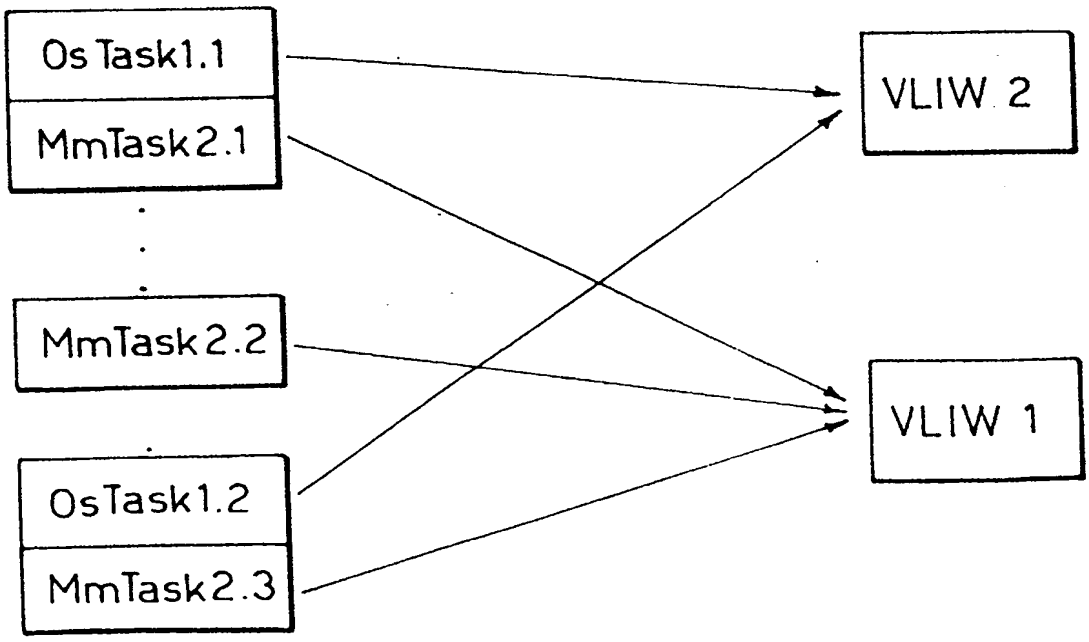
Os Task1.1
MmTask2.1
MmTask2.2
OsTask1.2
MmTask2.3
VLIW 2
VLIW 1

Fig_9

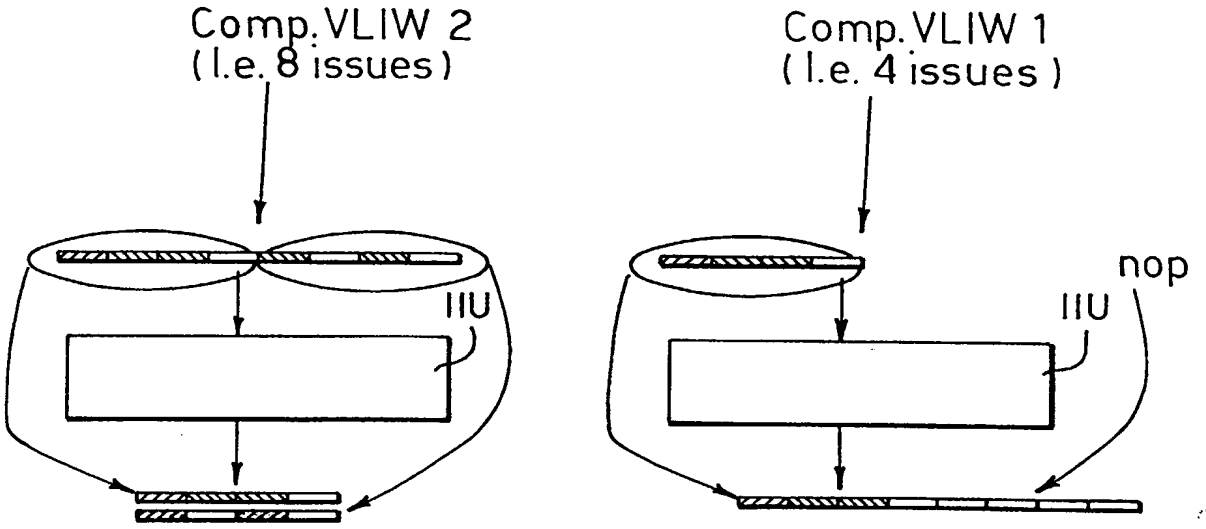
Comp.VLIW 2
(I.e. 8 issues)
Comp.VLIW 1
(I.e. 4 issues)
IIU
IIU
nop

Fig_10

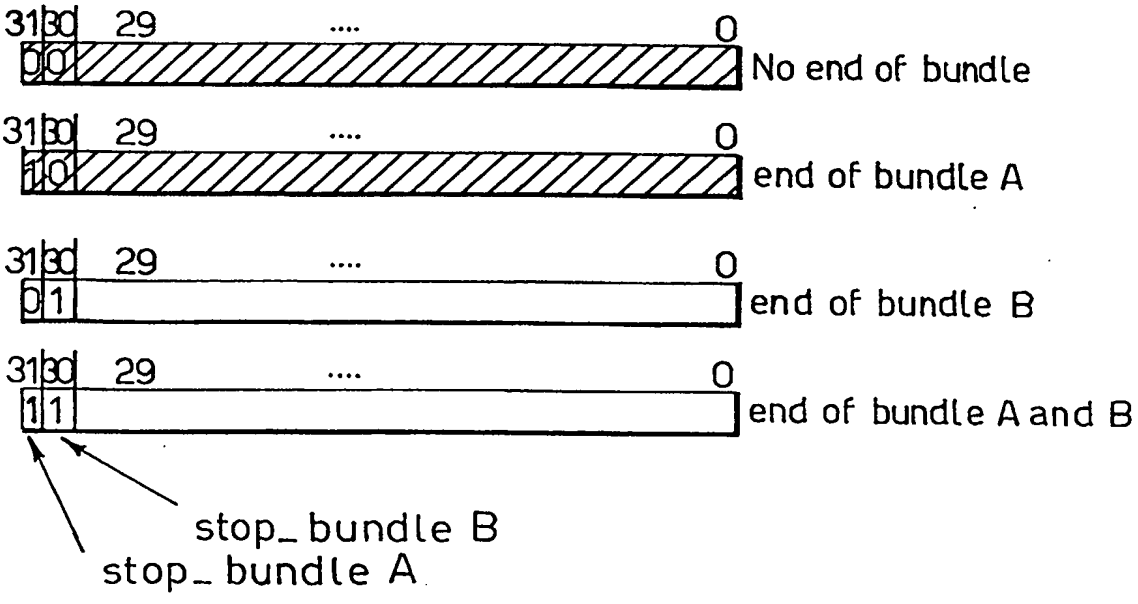
31 30 29 0
0 0
No end of bundle
31 30 29 0
1 0
end of bundle A
31 30 29 0
0 1
end of bundle B
31 30 29 0
1 1
end of bundle A and B
stop_bundle B
stop_bundle A

Fig_11

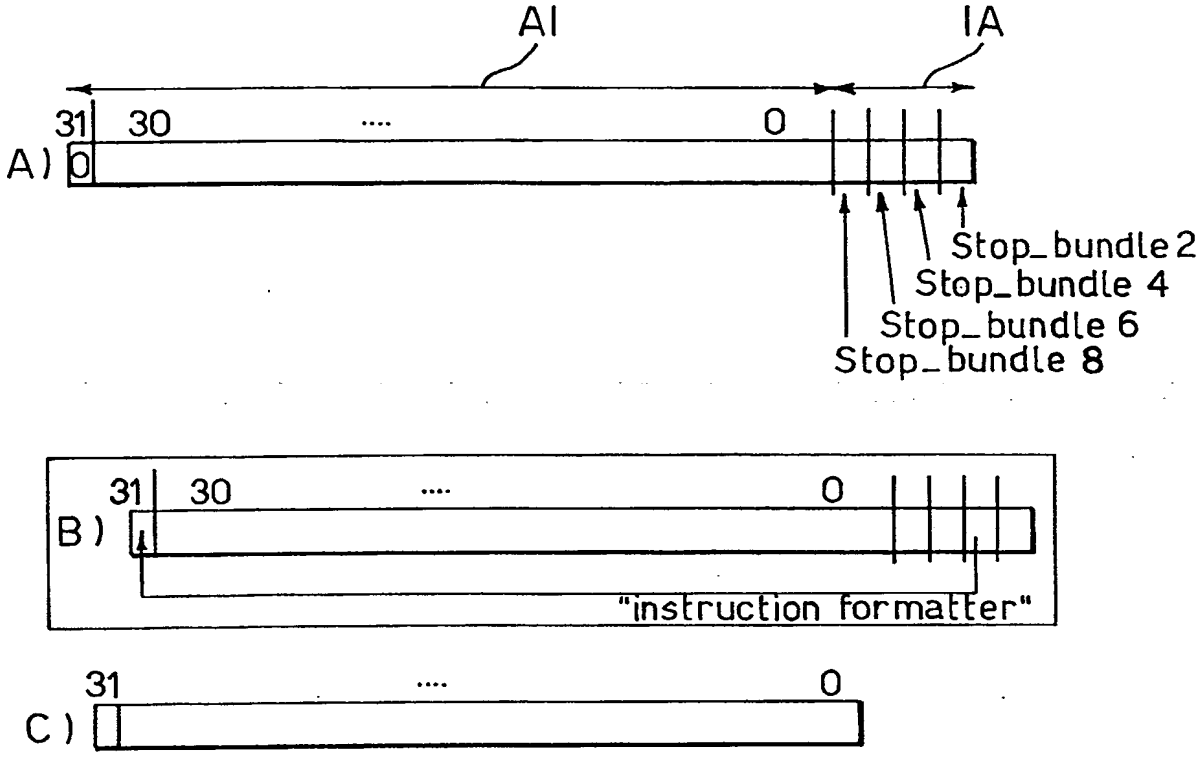
AI
IA
31
30
....
0
A)
Stop_bundle 2
Stop_bundle 4
Stop_bundle 6
Stop_bundle 8
B)
"instruction formatter"
C)

Fig_12

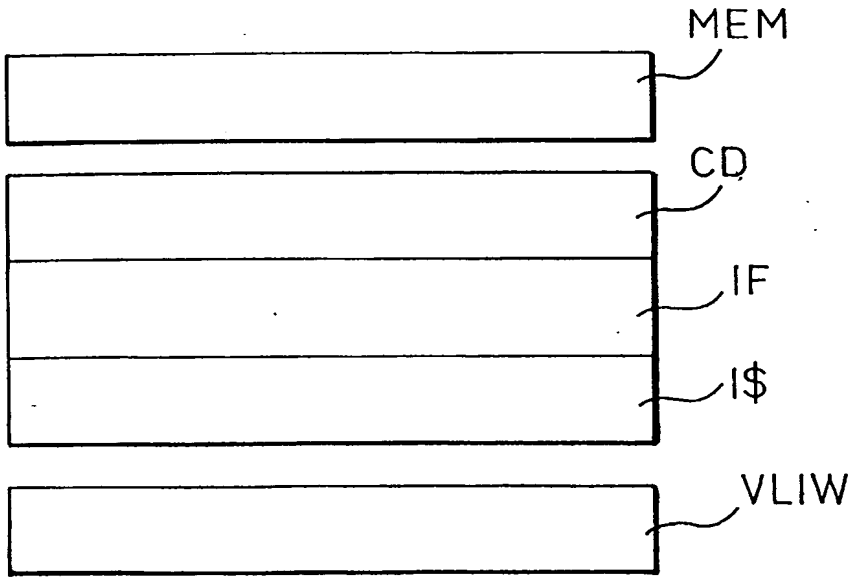
MEM
CD
IF
I$
VLIW

Fig_13

Process	Num.	% VLIW	Exec. time	Memory	Compiled For VLIW Of Length	Execution On VLIW Of length	Memory address
SystemIdleProcess	0	52	4:18:39	16K	4	4	x1234123A
System	2	02	0:00:37	200K	8	8	x3E586321
SMSS.EXE	26	00	0:00:00	0K	4	8	xFFC0A867
CSRSS.EXE	34	00	0:00:06	908K	4	4	xAFE25432
WINLOGON.EXE	40	00	0:00:01	32K	8	4	xCECC4223
SERVICES.EXE	43	00	0:00:01	1044K	8	4	

RIVENDICAZIONI

1. Procedimento per eseguire programmi su un sistema multiprocessore comprendente una pluralità di processori (VLIW1, VLIW2) aventi una data architettura di insieme di istruzioni (ISA), ciascuno di detti processori essendo suscettibile di eseguire, a ciascun ciclo di elaborazione, un rispettivo numero massimo di istruzioni,

caratterizzato dal fatto che comprende le operazioni di:

- compilare, almeno in parte, le istruzioni di detti programmi come parole di istruzione di lunghezza data (L1 risp. L2) eseguibili su un primo processore (VLIW1 risp. VLIW2) di detta pluralità,

- modificare almeno alcune di dette parole di istruzione di lunghezza data trasformandole in parole di istruzione modificate eseguibili su un secondo processore (VLIW2 risp. VLIW1) di detta pluralità, detta operazione di modificare comprendendo a sua volta almeno un'operazione scelta nel gruppo costituito da:

- suddividere dette parole di istruzione in parole di istruzione modificate, e

- inserire nelle parole di istruzione modificate istruzioni nulle (nop).

2. Procedimento secondo la rivendicazione 1, caratterizzato dal fatto che comprende le operazioni di:

- compilare le istruzioni di detti programmi in parte come prime parole di istruzione aventi una prima lunghezza data (L1 risp. L2) ed eseguibili su detto primo processore (VLIW1 risp VLIW2) di detta pluralità ed in parte come seconde parole di istruzione di lunghezza data (L2 risp. L1) eseguibili su un secondo processore (VLIW2 risp. VLIW1) di detta pluralità,

- modificare almeno alcune di dette prime parole di istruzione in prime parole di istruzioni modificate eseguibili su detto secondo processore (VLIW2 risp. VLIW1) di detta pluralità, e

- modificare almeno alcune di dette seconde parole di istruzione in seconde parole di istruzioni modificate eseguibili su detto primo processore (VLIW1 risp. VLIW2) di detta pluralità.

3. Procedimento secondo la rivendicazione 2, caratterizzata dal fatto che dette prime parole di istruzione e dette seconde parole di istruzione hanno rispettivamente una prima (L1) ed una seconda (L2) lunghezza massima con detta prima lunghezza massima (L1) maggiore di detta seconda lunghezza massima (L2), il quoziente fra detta prima lunghezza massima (L1) e detta seconda lunghezza massima (L2) avendo un valore dato (A) con eventuale presenza di un resto (B) e dal fatto che il procedimento comprende le operazioni di:

a) per modificare dette prime parole di istruzione aventi detta prima lunghezza massima (L1) in prime parole di istruzioni modificate aventi detta seconda lunghezza massima (L2):

- suddividere dette prime parole di istruzione in un numero di dette prime parole di istruzione modificate pari al valore di detto quoziente (A), e

- in presenza di detto resto (B), aggiungere a dette prime parole di istruzione modificate un ulteriore parola di istruzione modificata di lunghezza pari a detta seconda lunghezza massima (L2), detta seconda lunghezza massima essendo ottenuta inserendo in detta ulteriore prima parola di istruzione modificata un insieme di istruzioni nulle (nop), e

b) per modificare dette seconde parole di istruzione aventi detta seconda lunghezza massima (L2)

in seconde parole di istruzioni modificate aventi detta prima lunghezza massima (L1):

- aggiungere a dette seconde parole di istruzione di detta seconda lunghezza massima (L2) un numero di istruzioni nulle (nop) pari alla differenza (L1-L2) fra detta prima lunghezza massima (L1) e detta seconda lunghezza massima (L2).

4. Procedimento secondo la rivendicazione 1, caratterizzato dal fatto che comprende le operazioni di:

- codificare dette istruzioni su un numero dato di bit (N), detto numero di bit comprendendo un primo bit identificatore di una lunghezza di parola di istruzione eseguibile su un processore (VLIW1, VLIW2) di detta pluralità:

- associare a detto numero dato di bit, una rispettiva appendice (IA) comprendente un insieme di ulteriori bit identificativi di lunghezze di parole di istruzione eseguibili su diversi processori (VLIW1, VLIW2) di detta pluralità,

- identificare, per ciascuna di dette istruzioni, un processore di detta pluralità destinato ad eseguire detta istruzione, detto elaboratore identificato essendo suscettibile di elaborare per ciascun ciclo di elaborazione, una lunghezza di parola di istruzione data, e

- inserire nella posizione di detto primo bit identificatore un bit scelto fra detti ulteriori bit di detta appendice (IA), detto bit scelto essendo identificativo della lunghezza di parola di istruzione suscettibile di essere eseguita da detto processore identificato.

5. Procedimento secondo la rivendicazione 4, caratterizzato dal fatto che comprende l'operazione di

cancellare detta rispettiva appendice (IA) prima dell'esecuzione dell'istruzione.

6. Procedimento secondo la rivendicazione 4 o la rivendicazione 5, caratterizzato dal fatto che detto bit scelto viene inserito nella posizione di detto primo bit identificatore in una fase scelta fra:

- la decodifica dell'istruzione in vista dell'esecuzione,
- il refill della cache (I$) associata a detto processore identificato, e
- decompressione dell'istruzione in vista dell'esecuzione.

7. Procedimento secondo una qualsiasi delle precedenti rivendicazioni, caratterizzato dal fatto che comprende l'operazione di:

- distribuire alternativamente l'esecuzione delle istruzioni di detta sequenza fra i processori di detta pluralità (VLIW1, VLIW2, ...) dette istruzioni essendo direttamente eseguibili da parte dei processori di detta pluralità (VLIW1, VLIW2) in condizioni di compatibilità binaria.

8. Procedimento secondo la rivendicazione 7, caratterizzato dal fatto che comprende l'operazione di distribuire selettivamente l'esecuzione di dette istruzioni tra i processori di detta pluralità (VLIW1, VLIW2) distribuendo dinamicamente il carico computazionale di detti processori

9. Procedimento secondo la rivendicazione 8, caratterizzato dal fatto che comprende l'operazione di distribuire selettivamente l'esecuzione di dette istruzioni fra detti processori di detta pluralità (VLIW1, VLIW2) con il criterio di equalizzare la frequenza operativa dei processori di detta pluralità (VLIW1, VLIW2, ...).

10. Procedimento secondo una qualsiasi delle rivendicazioni 7 a 9, caratterizzato dal fatto che comprende l'operazione di svolgere un processo di controllo eseguito da almeno uno dei processori di detta pluralità così da equalizzare il proprio carico di lavoro rispetto agli altri processori di detto sistema multiprocessore.

11. Procedimento secondo la rivendicazione 10, caratterizzato dal fatto che comprende l'operazione di costituire una tabella accessibile da detto processo di controllo, detta tabella comprendendo voci scelte nel gruppo costituito da:

- una lista di processi (Process) in esecuzione o sospesi su un qualsivoglia processore didetta pluralità,
- il numero progressivo (Num) dello stesso in base all'ordine di attivazione,
- la percentuale (% VLIW) di potenza massima del processore che è utilizzata da detto processo,
- il tempo di esecuzione (Exec.time), detto tempo, se nullo, indicando che il processo è temporaneamente sospeso dall'esecuzione,
- la quantità di memoria (Memory) del sistema utilizzato dal processo per poter eseguire la funzione al quale è preposto,
- la lunghezza massima (Compiled For VLIW Of Length) dell'istruzione lunga che il processore VLIW può eseguire e per il quale era stata generata durante la compilazione,
- lunghezza massima (Execution on VLIW of Length) dell'istruzione lunga del processore VLIW sul quale è eseguito, e
- l'indirizzo della porzione di memoria (Memory address) nel quale i dati e le istruzioni sono memorizzate.

12. Sistema multiprocessore, configurato per operare con il procedimento secondo una qualsiasi delle rivendicazioni 1 a 11.

13. Sistema multiprocessore secondo la rivendicazione 12, caratterizzato dal fatto che detti processori sono tutti del tipo VLIW.

14. Sistema multiprocessore secondo la rivendicazione 12, caratterizzato dal fatto che detta pluralità di processori comprende almeno un processore VLIW ed almeno un processore superscalare.

RIASSUNTO

Programmi aventi una data architettura di insieme di istruzioni (ISA) sono eseguiti su un sistema multiprocessore comprendente una pluralità di processori, ad esempio di tipo VLIW, ciascuno di tali processori essendo suscettibile di eseguire, a ciascun ciclo di elaborazione, un rispettivo numero massimo di istruzioni. Le istruzioni sono compilate come parole di istruzione di lunghezza data (L1 risp. L2) eseguibili su un primo processore (VLIW1 risp. VLIW2). Almeno alcune delle parole di istruzione di lunghezza data sono trasformate (IIU) in parole di istruzione modificate eseguibili su un secondo processore. L'operazione di modificare comprende a sua volta almeno un'operazione scelta nel gruppo costituito da:

- suddividere le parole di istruzione in parole di istruzione modificate, e
- inserire nelle parole di istruzione modificate istruzioni nulle (nop).

(Figura 9)